

Projet de Programmation Orienté Objet : Chat

6 Février, 2021

Étudiants :

Hugo LE BELGUET lebelgue@etud.insa-toulouse.fr

Dany LAU dlau@etud.insa-toulouse.fr

Tuteurs :

Sami Yanguï
Hendry F.Chame

Plan

Introduction

Conception Orientée Objet

Diagramme des cas d'utilisation

Diagramme de séquence

Diagramme de classes

Diagramme de structure composite

Diagramme de machines à états

Programmation Orientée Objet

Explication du code et choix techniques

Guide d'installation

Guide d'utilisation

Perspective d'amélioration

Conclusion

Introduction

Dans ce projet nous allons aborder dans un premier temps la partie conception puis dans un deuxième temps la partie programmation. Au cours de la conception le but est de visualiser le projet, les contraintes, ce que nous allons devoir délivrer pour respecter au mieux le cahier des charges. Chaque diagramme réalisé a pour but d'explorer une nouvelle facette et de construire un peu plus le projet pour savoir dans quelle direction aller lors de son implémentation.

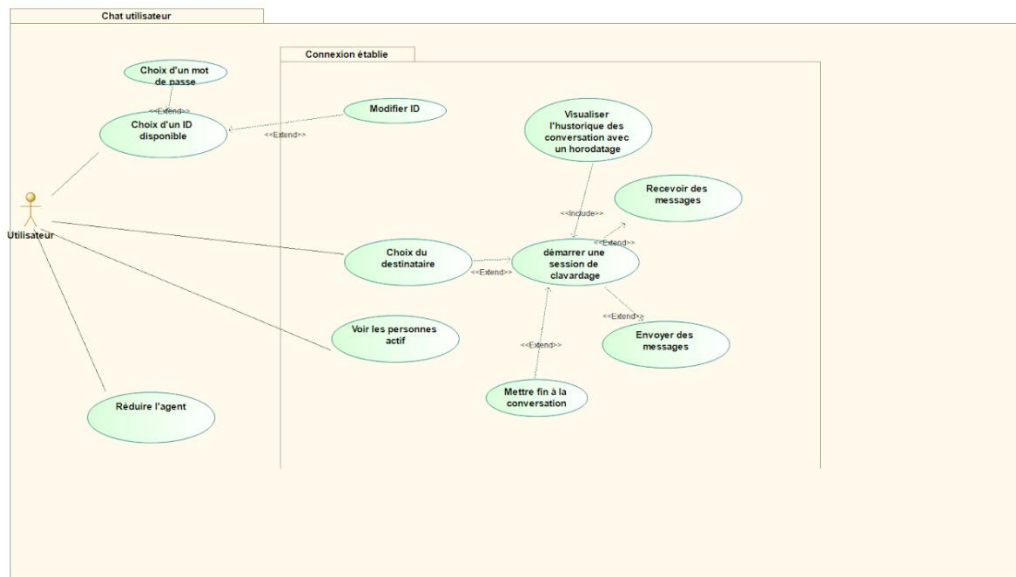
La suite de ce rapport traitera de la partie réalisation, nous nous attarderons sur les différents choix techniques que nous avons fait, en plus d'expliquer de manière concise et bref le fonctionnement de chaque classe.

Conception Orienté Objet

Diagrammes des cas d'utilisation :

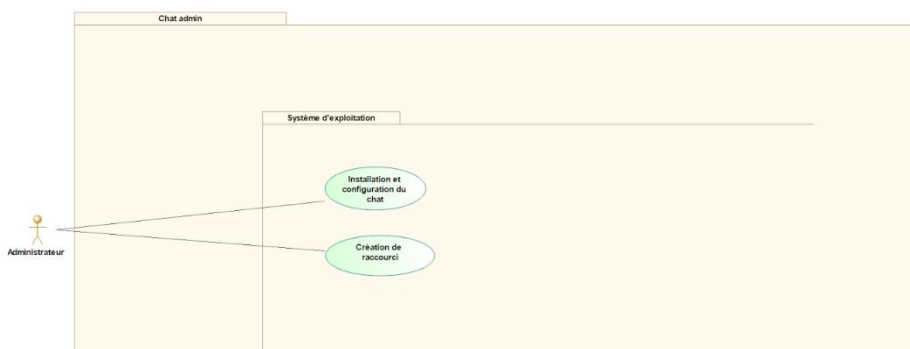
Pour commencer, au travers de ces deux diagrammes nous voulons présenter les différentes façons dont l'utilisateur va pouvoir interagir avec le logiciel. Ici on observe donc qu'il aura plusieurs possibilités, il pourra réduire la fenêtre du logiciel, choisir un ID qui sera visible par tous les utilisateurs ainsi que le changer, il pourra voir les personnes actives sur le réseau et finalement, choisir un des utilisateurs pour tchater avec lui.

Diagramme de cas d'utilisation de l'utilisateur :



L'administrateur de son côté n'aura qu'un rôle d'installateur, il sera chargé de mettre en place le logiciel.

Diagramme de cas d'utilisation de l'administrateur :



Le diagramme de séquence permet de décrire comment et dans quel ordre plusieurs objets fonctionnent ensemble. Dans notre cas nous en avons réalisé deux, le premier illustre la phase de connexion d'un utilisateur ainsi que l'échange de message avec un autre utilisateur situé sur le même réseau jusqu'à sa déconnexion du service. Le deuxième se concentre plus particulièrement sur le changement de pseudo et la façon dont c'est géré vis-à-vis des autres utilisateurs car un pseudo doit être unique.

Diagramme de séquence :

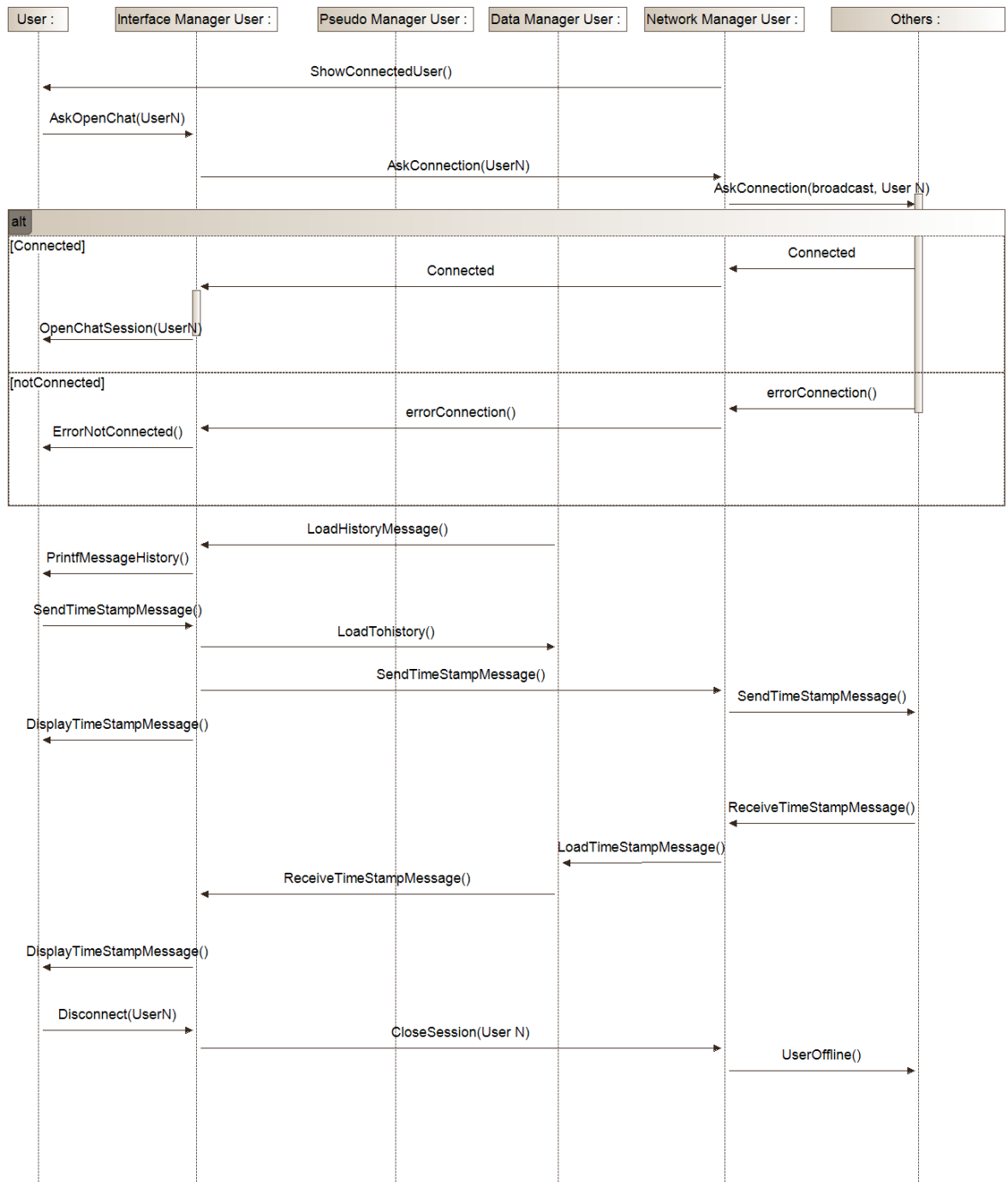


Diagramme de séquence changement de pseudo :

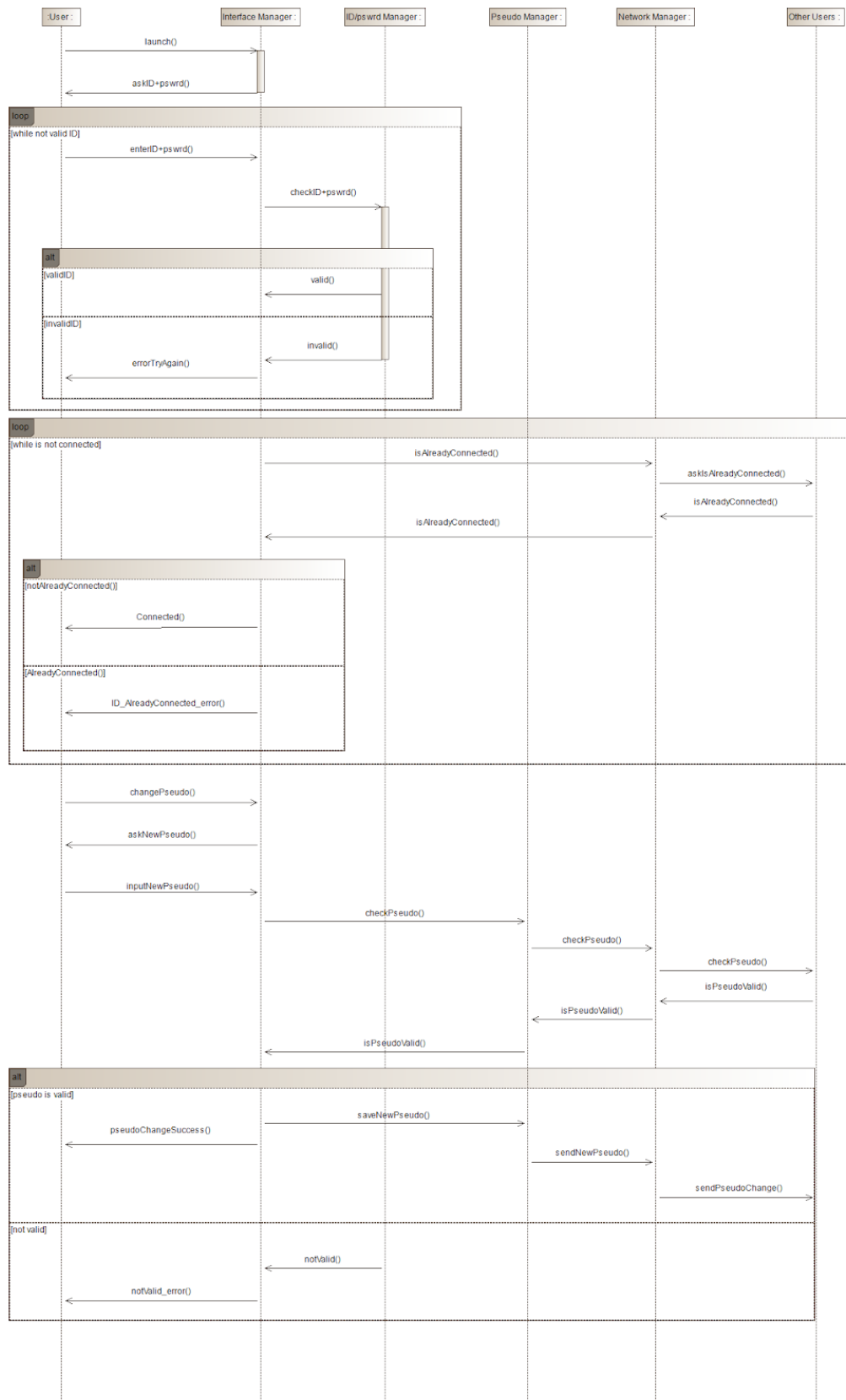
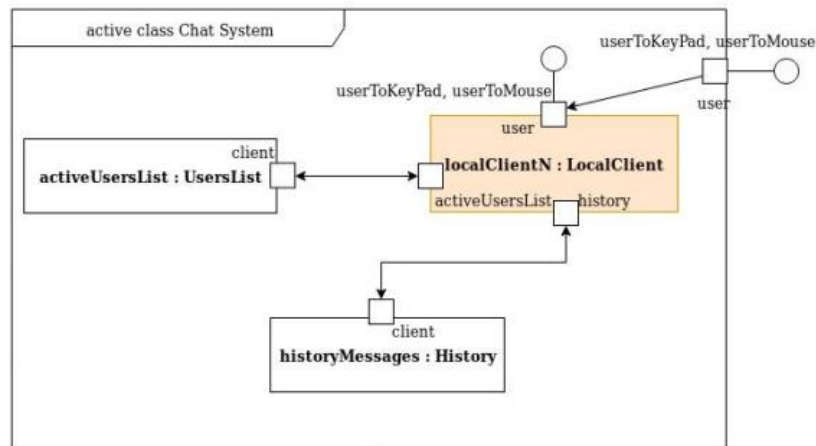


Diagramme de classes :



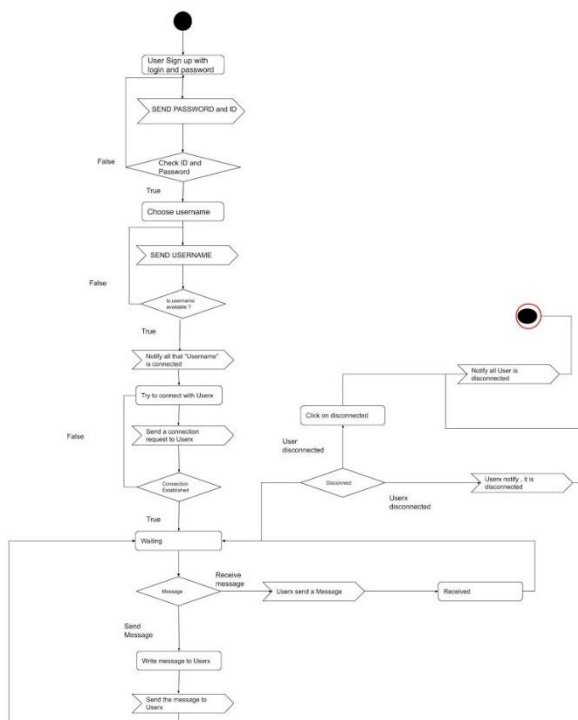
Le diagramme de structure composite permet de donner une vue d'ensemble du logiciel.

Diagramme de structure composite :



Un diagramme de machine à état permet de montrer le cycle de vie des différents objets de classe d'un logiciel, on peut y observer les possibilités selon les cas.

Diagramme des machines à états :



Programmation Orientée Objet :

Explication du code et choix techniques :

Dans cette partie nous allons parcourir le code réalisé en expliquant ce qui a été fait, pourquoi ça a été fait et le résultat attendu.

La classe « Main » ne sert qu'à créer la fenêtre connexion, c'est la première étape de notre programme, ce qui lance tout le reste.

Nous allons démarrer avec la première classe, « swingconnection ». On commence par créer une liste avec des string pour créer une paire « user/mdp » identifiée par des indices respectivement $i/i+1$. On a ensuite un listener sur le bouton « se connecter » qui va tester ce que l'utilisateur a rentré avec ce que contient la liste, si la paire rentrée correspond à la paire de la liste la connexion est autorisée.

La classe « swingpseudo » fonctionne sensiblement de la même manière, à la différence que le seul test consiste à vérifier que la zone de texte ne soit pas vide au moment de cliquer sur le bouton, le serveur s'exécute ensuite.

Nous allons maintenant voir le fonctionnement des communications UDP dans un premier temps. Pour commencer la classe « Udpserve », elle écoute en permanence ce qui arrive, traduit ce qu'elle reçoit en string puis teste si les données reçues sont déjà dans la liste des pseudos de la classe pseudonyme, si ça n'est pas le cas, elle y ajoute le pseudo. « Udpenvoie » de son côté est chargée d'envoyer le pseudo ainsi que l'adresse IP de chaque utilisateur avec une syntaxe précise, et ce toutes les cinq secondes. Nous avons choisi d'utiliser des threads car il y a beaucoup d'attente active (while(true)), donc beaucoup d'éléments bloquant pour le programme, ça permet également une structure plus solide ainsi qu'un meilleur contrôle des ressources et une organisation plus claire.

Le fonctionnement de notre serveur TCP est un peu particulier, nous avons décidé d'ouvrir et de fermer la connexion client/serveur à chaque échange de message. C'est-à-dire que quand un client veut joindre un serveur, une connexion va s'établir, le message va être envoyé et le serveur va immédiatement fermer la connexion. Cela permet d'avoir un meilleur contrôle du flux, de savoir à tout moment quel est le statut de connexion, s'il est ouvert ou fermé. Le client n'a rien à dire pour fermer la connexion, ce qui évite d'avoir une fenêtre restée ouverte en permanence si le client crash sans avoir demandé la fermeture de la connexion. Ce système consomme un peu plus de ressources mais permet une meilleure gestion et est adapté à ce type de communication.

De son côté la classe externe « Chatnotify » sert d'observer, quand on reçoit un message la classe ajoute le message et l'id de l'utilisateur, ce qui permet de l'afficher dans la fenêtre de discussion. Cette méthode évite les « null pointer exception » et c'est la dernière classe créée avant la création des serveurs, elle possède également tous les arguments des autres classes.

Le but principal de la classe « Chat » est de récupérer les messages et le pseudo correspondant pour ensuite l'afficher sur la fenêtre de discussion. Cette classe est assez complexe, elle commence par une liste de string pour chaque user, et une liste de liste regroupant tous les users, chaque élément sera initialisé avec « unknown ». Cette méthode est assez limitante

pour le programme mais c'est dû au fait que nous n'avons pas trouvé de moyen de créer des variables à nom dynamique. Les trois classes suivantes consistent, dans un premier temps, à vérifier si l'utilisateur a déjà été ajouté dans une liste, si non l'ajouter, et ensuite mettre dans la liste de chaque utilisateur les messages qui lui sont destinées pour son affichage. De même il y a exactement les mêmes fonctions mais cette fois dans un but purement local pour afficher les messages sur sa propre interface.

La classe « Pseudonyme », comme son nom l'indique, permet de gérer, ajouter, afficher les pseudos des utilisateurs ainsi que le propre pseudo de l'utilisateur. Il y a également une méthode pour retourner l'adresse IP de l'utilisateur et une autre pour sélectionner un utilisateur en particulier dans la liste selon son ID.

Notre dernière classe « swingchat » va gérer tous les boutons de l'interface chat à l'aide de cas. Cette classe contient également l'iduser qui va agir comme un fil rouge entre toutes les classes pour identifier l'utilisateur à qui on souhaite s'adresser. Le dernier rôle de cette classe est d'envoyer les messages à l'adresse IP choisi lors de la sélection de l'utilisateur.

Guide d'installation :

Etape 1: Téléchargement de l'application sur GitHub

Aller sur GitHub et télécharger le zip (POOHugoLeBelguetDanyLauIRSC.zip).

Lien Github : <https://github.com/hugo-insa/COO-Hugo-Dany.git>

Etape 2: Ouvrir Eclipse et importer le projet

Dézipper le dossier.

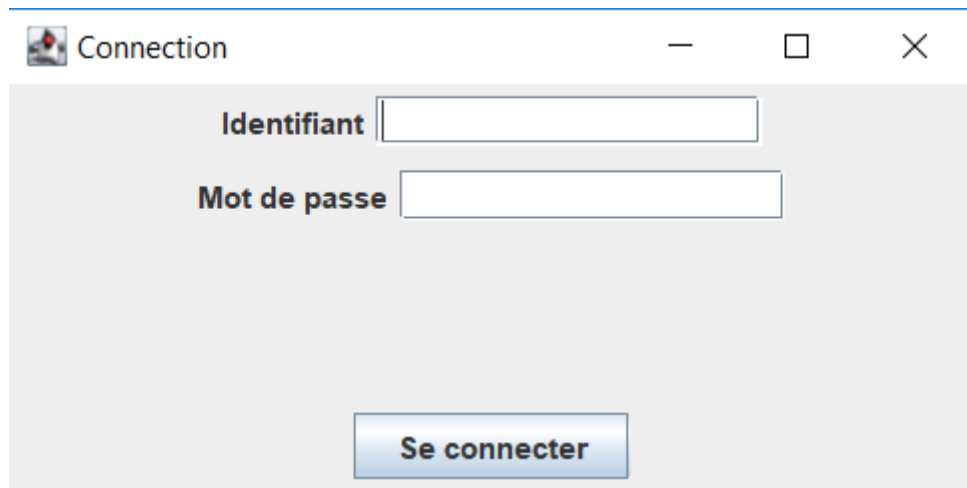
Aller dans File -> Open Projects from file system... puis sélectionner le dossier et enfin finish.

Etape 3: Lancer l'application

Pour lancer l'application, il suffira d'exécuter la classe main en tant qu'application Java.

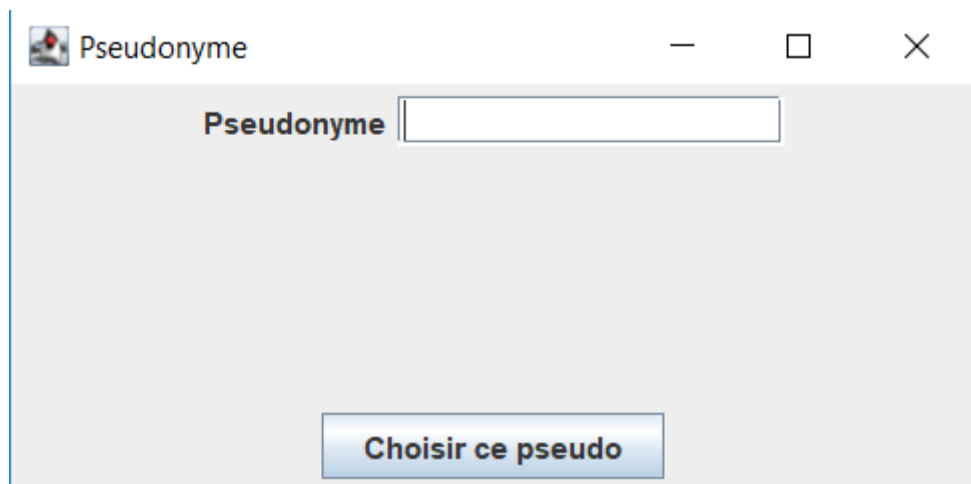
Guide d'utilisation :

Lorsque toutes les étapes d'installation ont été effectuées, et que l'application a été lancée, la première fenêtre qui s'ouvre est la fenêtre de connexion :



Voici deux jeux d'identifiants/mots de passe pour avoir accès à l'application : hugo/hugo et dany/dany. D'autres sont disponibles (user1/user1 et user2/user2).

Une fois connecté on arrive sur la fenêtre qui nous demande de choisir un identifiant :



Ici, on peut rentrer n'importe quel pseudonyme (dans la limite des caractères autorisés à savoir 15), c'est ce que verront les autres utilisateurs.

La prochaine fenêtre est celle du chat en lui-même. On y trouve quatre boutons tous utiles pour communiquer.

Le premier, « Users » permet d'afficher la liste des utilisateurs actuellement connectés, cet affichage comprends un numéro, le pseudo de l'utilisateur et son adresse IP.

Le deuxième, « Select » permet de choisir l'utilisateur avec qui on souhaite discuter, pour ce faire on écrit le numéro de cet utilisateur dans la barre de texte puis on fait select.

Le troisième, « Send » permet d'envoyer le message tapé dans la barre de texte à l'utilisateur précédemment sélectionné. Par défaut c'est le premier utilisateur de la liste qui est sélectionné.

Le dernier bouton, « Refresh » permettait d'actualiser le chat pour voir les messages envoyés ou reçus. Néanmoins cette fonctionnalité a été implémenté par la suite les messages s'affichent maintenant dynamiquement sans intervention. Nous avons laissé le bouton au cas où il s'avère nécessaire.

Pour utiliser le chat en local sur la même machine il faudra changer les ports dans les classes « Udpserve » ligne 16, « Udpenvoi » ligne 27, « Serveur » ligne 37 et dans « swingchat » ligne 100. Autrement la meilleure façon de tester le programme reste encore d'utiliser deux ou plus ordinateurs différents sur le même réseau.

Lors de la première utilisation, il faut attendre 5 secondes avant d'utiliser le bouton « users » pour que la liste s'affiche.

Perspective d'amélioration :

Notre programme peut être améliorer, notamment en rajouter une gestion de base de données pour les messages afin d'avoir un historique qui perdure dans le temps plutôt que ce que nous avons actuellement. De plus notre programme ne supporte pas les utilisateurs qui ne sont pas sur le même réseau et il ne permet pas de changer de pseudo malgré la présence de classe « changerpseudo » qui promet des perspectives d'amélioration. On peut également imaginer par la suite une gestion d'envoi de fichiers et images.

Conclusion

Ce projet nous a permis de voir toutes les étapes par lesquels on doit passer lors de la réalisation d'un projet complexe avec un cahier des charges précis et assez complet. C'était intéressant de se rendre compte qu'une bonne partie du projet ne consiste pas seulement en de la programmation, mais qu'il faut d'abord passer par de la conception et surtout l'importance de cette partie pour avoir une vision claire du projet et savoir dans quelle direction partir lors de la programmation. C'est un projet que nous avons trouvé malgré tout peut être un peu trop complet, nous n'avons pas été en mesure de répondre à toutes les attentes, mais ce que nous retenons tout particulièrement, et ce pour quoi nous nous sommes le plus appliqué, c'est la façon de coder en Java, pour avoir un code extrêmement clair, concis, c'est-à-dire que nous avons limité au maximum la longueur pour le rendre intelligible et pas rébarbatif à lire. De plus nous avons fait en sorte de respecter toutes les normes et bonnes mesures propres au Java.